Developing the Virtual Communicator AI backend of a chatbot

By: Arijit Pramanik
Mentor: Dr.Rajendra Singh Sisodia
Philips Innovation Campus, Bangalore

08 July 2017

Abstract. The chatbot is based a information retrieval system which specialises in domain knowledge and is well-acquainted with general conversations also.

1. Aim of the Project

The objective of this project is to build a conversation engine, which is supposed to be acquainted with various queries from the consumers related to a particular product. Our aim is to build the *querying system* based on the user manual of the respective product.

2. Use Case

The principal use case employed for testing is the **Sonicare Diamond Clean 300 Series** toothbrush manual, though other manuals related to the air-fryer and Philips shaver have also been tried out.

3. Query System

3.1. Training the chatbot

First of all, the chatbot is trained on four *corpuses*, namely being:

- Output.txt (a stemmed version of the manual with each sentence on a new line and devoid of any punctuation and stopwords)
- Topic1.txt (file which contains the list of all topics, stemmed and devoid of punctuation and stopwords corresponding to the list of all relevant topics with each one on a new line)
- Text8Corpus (This has been stemmed and stripped off stopwords and is a general text Corpus)

- BrownCorpus (This is NLTK's general text corpus which also has been thoroughly stemmed and stripped off stopwords)

The parameters need to be manually fine-tuned, namely size (*set to the total number of topics*), window is the distance upto which other adjacent words should be considered to be in the same context. The learning rate and number of epochs may be set to the default value itself. 'min_count' indicates the minimum number of occurrences of a word to be included into the vocabulary.

3.2. Preprocessing of question

Any question entered via the user interface, is stemmed (*refer to SnowBall Stemmer*) and stopwords are removed (*stop.txt is the list of stopwords).* All punctuation is removed.

The words are then corrected in case of spelling errors (*big.txt is intended for this purpose for the list of domain-related words*)

You can include special words which will not be stemmed or corrected for spelling errors in *donot.txt*

3.3. Finding the most relevant topic

A list of topicsis passed by the knowledge engine and the most relevant topic is selected by the algorithm. It enters yet another sublist of topics under the former and same is recursively applied until the most relevant topic (or subtopic) is found.

The response from the knowledge base might be the answer pertaining to the relevant topic found or a list of subtopics out of which there is no clear answer or a response declaring the irrelevance of the question to the given list of topics.

A **Sentiment Analyser** deployed, checks for the user satisfaction of the answer and returns the next most relevant response pertaining to the user's question.

The chatbot has limited domain capabilities and it reroutes the question to an AIML engine which engages with the user in real-time in case the question is not restricted to the domain.

3.3.1. Semantic matching **Gensim** library is used for initialising the *word2vec* model, where each word is converted to a vector form. In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. The bag-of-words model is commonly used in methods of document classification.

Tf-idf Vectorisation is useful for marking the important (less-frequent words) without having any bias for longer documents. The **inverse document frequency**

is a measure of how much information the word provides, that is, whether the term is common or rare across all documents. **Term frequency** is the raw count of a term in a document. This gives weights to the words and hence a corresponding normalized vector is generated which works better than the boolean vectors in bag-of-words model.

Latent Semantic Indexing is a technique used for analyzing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. LSA assumes that words that are close in meaning will occur in similar pieces of text.

Latent Dirichlet Allocation is a generative probabilistic model for collections of discrete data such as text corpora. LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics. Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities. In the context of text modeling, the topic probabilities provide an explicit representation of a document.

3.3.2. Similarity measures The similarity measures employed work on the underlying dot product of the vectors representing the words. Thus, this is the cosine similarity of two words. Using *n_.similarity* of gensim, we apply this measure between the question, and the list of topics at that hierarchical level. The returned list is sorted by this similarity value.

Other measures include the *Jaccard distance*, *Word Mover's Distance*. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets. WMD is a method that allows us to assess the "distance" between two documents in a meaningful way, even when they have no words in common. It uses word2vec vector embeddings of words. Both of these were implemented and the results led to lesser accuracy than the cosine similarity measure.
The similarity measure of gensim doesn't work well for synonyms, but well enough for cases like: 'woman' + 'king' - 'man' = 'queen'

Alternative libraries like Spacy also have a similar feature of detecting relationships in a better fashion, but is outwitted by gensim's optimisation of using C++

3.4. Analysis

Three models have been implemented so far to reply to consumer queries.

(i) **QnA** : The first model simply works on a **Tfidf model**. The list of topics from the knowledge base at the specified hierarchical level are transformed via a vector-to-keyword mapping after being pre-processed (*stripped off stopwords and stemmed, while preserving proper domain words like 'sonicare' and 'philips'*). Then, the tfidf transformation is applied. The same preprocessing and transformation occurs to

the question, resulting in a vector. The cosine similarity measure between the vectors helps us list out the most relevant topics.

(ii) **QnA2** : The second model is a combination of four models. Both **LSA** and **LDA** models are employed. The first two models are trained on the *pre-processed list of topics* while the latter **LSA** and **LDA** models are trained on a *pre-processed corpus of the manual* and the *list of all relevant topics in the manual*. The question is pre-processed (textitlemmatized) and ***MatrixSimilarity*** of the in-built similarities class is used to index the pre-processed query and return the list of relevant topics accordingly.
The corpus of the manual, list of topics and the question are all subjected to the **Tf-idf transformation**, **LSI** and **LDA transformations** on the former and the latter.

(iii) **QnA3** : The **word2vec model** of gensim has been trained on all four corpuses mentioned. The model is initialised with the *manual text* via the LineSentence constructor, and further vocabulary and training is done on the other models. The query, pre-processed as before is passed to gensim's n_similarity function which in turn ranks the topics in order of their relevance scores.
The "$" functionality can be used to resolve contexts by emphasizing on a particular set of words that are believed to bring clarity to the context. Please just add a "$" symbol as prefix to the words that are deemed important.

3.5. Reflections and Future Work

The last model in general, has greater accuracy and points to the incorrect answer in case the keywords lie deep in the hierarchical model. This can be improved if the XML document is made more carefully, under supervision of domain experts. The second model gives relatively lower scores than the former, and hence the thresholds have to be readjusted. The first model works relatively well, but only when keywords are present in the question.

Dynamic thresholding can be used to resolve clear conflicts in deciding the most relevant topic. Currently, these are hardcoded values. Moreover, the XML document should be created in a well-formatted style so that traversing the tree is well-guided by the topic headers.

3.5.1. Synonym Engine Various libraries are out there in python which can give words similar to a particular word. **Spacy** provides a list of most similar words which can be filtered accordingly as per the vocabulary available. Other available packages are Wordnet, Pydictionary.

4. Sentiment Analyser

The library that has been deployed is **NLTK's vader**. It has already been pre-trained on movie reviews, semantic lexicons and emoticons. The **polarity_score** returns a dictionary of compound, positive, negative and neutral values. If the compound value is greater than 0.5, then it carries a positive sentiment. If it is less than -0.5, then it carries a negative sentiment. Else, it is classified as neutral.

4.1. Emotions Classifier

The list of emotions present are the *Ekman's* six basic emotions : *Happiness, Sadness, Surprise, Fear, Disgust, Anger* and an additional emotion, *Love*; except the fact that *Interest* is included instead of *Disgust*.

The corpus essentially consists of movie reviews found online and have been annotated for the emotions they exhibit. Each review is followed up by a list of _ seperated emotions while being followed by *None* if there are no emotions exhibited. The set of emotions is automatically gathered from the reviews. One annotated review from the corpus is listed for reference:
"There is some spectacular, heart stoppingly beautiful photography here of a range of scenery and animals, from arctic to tropical and everything in between.->Love_Surprise"

Given a suitable corpus, to effectively split it for training and testing, run ***split-train_test.py*** to test the accuracy of the classifier, which given the current corpus of 500 reviews arising from 7 movies, stands at 40.4%. The metric used for assessing this is scikit-learn's *metrics.accuracy_score*.

The classifier used for this approach is a **OneVsRestClassifier**. The estimator for the above is a **LinearSVC** Since each review can have multiple labels, this is a *multi-label, multiclass* classification problem. We use a **MultiLabelBinarizer** to transform the labels of a review into a boolean vector. A self-written functon, arijit_score is being used as a measure of false positives returned by the classifier. **TfidfVectorizer** is used on the training and test data to remove all stop words, and an ngram-model is used which resorts to unigrams and bigrams only.

GridSearchCV in scikit-learn can be used as a tool to find out the parameters for a given classifier that best fits the data. Currently, the best-fit has been obtained by fine-tuning it manually.

4.2. Sentiment Plotter

For tracking the sentiment changes across replies of a user in a chatbot-enabled response system, this can be a useful tool for the summary.

The chunk of text (review) is taken as input from a file and decomposed into sentences. Each sentence is tested for a sentiment value, which lies between -1 and 1. The immediate sentiment values are plotted with a black line and the cumulative sentiment values with a red line. If the red line peaks above the baseline, then the review is a positive one, else generally the red line sloping downwards is an indication of a negative review.

4.3. Analysis and Reflections

The accuracy of this classifier may be improved using other classifying estimators other than LinearSVC, like *Logistic Regression, RandomForestsClassifier* and *fine-tuning via GridSearchCV*.
The corpus should also be improved in terms of the number of available training instances from a reliable source of movie reviews.

5. Ontologies and RDF

To convert any .rdf file in RDF/XML format to N-Triples format and vice versa, use http://rdf-translator.appspot.com/ It also supports other formats like Turtle and N3.

The advantage of the RDF format is that we can use SPARQL to query with specified graph structure to match graph pattern of given RDF to find out the unknown variables (e.g. ?city). The queries are of four types:

(i) **SELECT** simply returns data matching some conditions (a graphical pattern)

(ii) **ASK** queries check if there is at least one result for a given query pattern. The result is true or false.

(iii) **CONSTRUCT** queries returns an RDF graph that is created from a template specified as part of the query itself.

(iv) **DESCRIBE** queries returns an RDF graph that describes a resource.

N-Triples format advantage : Is of the form Subject-Predicate-Object. This is Intuitive and similar to the subject, predicate and object of an english sentence. The three types of objects can be URIRef, Literals (Strings and integers) or BNodes (Blank nodes identified by any alphanumeric string beginning with _: with some restrictions) RDFLib is a useful python library for parsing RDF files.

RDF Validator (W3C) can be used to validate rdf documents in xml format : https://www.w3.org/RDF/Validator/rdfval (It can also be used to visualise the RDF graphs).

An illustration as to how blank nodes can be useful: SPARQL queries for blank nodes:

SELECT ?a ?b WHERE ?a :predicate [:otherPredicate ?b] .

equivalent to:

SELECT ?a ?b WHERE ?a :predicate _:blankNode . _:blankNode :otherPredicate ?b .

It is very difficult to convert N-Triples to english sentences and vice versa : NLTK parsers use subject-verb-object structure which is not very helpful. **Dependency parser of SpaCy** can be helpful in visualising the structure and dependencies of a sentence. You can analyse the intended subject, object and predicate of the sentence.

Stanford's Protege is a GUI based tool than uses the **OWL API** and can be used to create RDF files in an OWL format which is almost similar to RDF files and can be queried using SPARQL. OWL files are a bit compact compared to those in RDF/XML format. It requires an underlying base OWL file which serves as the template for the RDF file being created. We can create new properties as a subproperty of Thing in *Data Properties*. The objects in that file can be instantiated by creating new objects under the *Individuals* tab. Then, we can save it in the required format, preferably RDF/XML.

5.1. Exploring additional features

Some key plugins that might be useful in generating ontologies:

(i) XML2OWL : Can be used to generate ontologies out of XML files. Mappings are carried out according to user-provided rules, and result in new individuals and axioms in the ontology.

(ii) XML Tab : The XML Tab enables users to import an XML document into Protege, creating a set of classes and instances in a knowledge base which correspond to the entries in the XML document.

(iii) Excel Import : Import content and generate classes from Excel or CSV files. Provide rules to create restrictions between columns.

5.2. Making a sample ontology via Protege

- Firstly, you have to select some IRI for your ontology, some resource URL that you yourself own, e.g. http://localhost:9004/myontology/
- Secondly at the bottom of the screen, use the option import ontology, to select an ontology template to be used as the backbone framework, e.g. FOAF (friend of a friend) ontology can be imported from the web.
- Next, decide upon the prefixes which will appear adjacent to the 'Import Ontology' tab. For example, you may use foaf as https://xlmns.com/foaf/0.1/
- You can also create new data properties as subset of the property:Thing and specify its type, like its Range(e.g. Literal) and its type whether its an Agent(standard property type of FOAF)
- Now you can click on the Individuals tab at the top and create new individuals. Whenever you create one, don't forget to specify its type (e.g. Person), and other data properties like name, surname, birthday, skype ID, etc. (As occurs in FOAF)

- Lastly, save this in the format that you want using Save As. RDF/XML format is the default format that appears, but the file is created as per the OWL ontology format.

  For more details, refer to : https://www.youtube.com/watch?v=leO7__ZonbQ&t=9s

6. References

Github repositories referred to:

- https://github.com/josephwilk/semanticpy/tree/master/semanticpy (Code for model1)
- https://github.com/jpabbuehl/spark-n-spell/blob/master/symspell_python.py (A powerful offline domain spelling checker)
- https://github.com/cjhutto/vaderSentiment (Sentiment values from -1.0 to 1.0)
- https://github.com/NLeSC/spudisc-emotion-classification (Corpus for emotional classification)

  Few good tutorials :

- https://radimrehurek.com/gensim/tut1.html (Tutorials for gensim : Aseries of three)
- http://www.cambridgesemantics.com/semantic-university/learn-sparql (SPARQL tutorials)
- http://scikit-learn.org/stable/modules/multiclass.html#one-vs-the-rest (scikit-learn)
- http://www.nltk.org/howto/stem.html?_sm_au_=iVVSfFPJkqMSQVQH (Stemmers)
- https://radimrehurek.com/gensim/models/word2vec.html#id4 (Wordvec)

  A few good reads:

- https://arxiv.org/pdf/1301.3781.pdf
- https://groups.google.com/forum/#!topic/gensim/wwHVnbYLM6I
- https://www.aclweb.org/anthology/C/C08/C08-1111.pdf
- http://www.hlt.utdallas.edu/ kirk/publications/robertsLREC2012_2.pdf
- http://nvlpubs.nist.gov/nistpubs/ir/2015/NIST.IR.8068.pdf